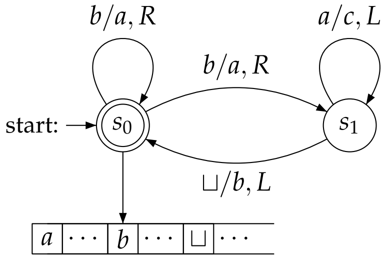
CS 208: Automata Theory and Logic

Part II: Turing Machines and Undecidability
Lecture 1: Turing Machines

S Akshay



.

Department of Computer Science and Engineering,
Indian Institute of Technology Bombay.

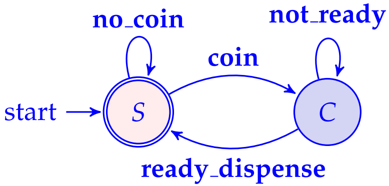
Outline of the lecture

In this lecture we will cover:

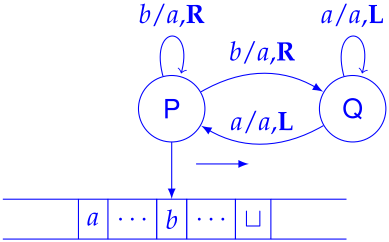
- Formal definition of Turing machines
- Examples of Turing machines
- Variants of a Turing machine
  - Multi-tape Turing machines
  - Non-deterministic Turing machines
- Running time of a Turing machine
- The Church-Turing thesis and its importance to computer science

From finite to infinite memory

Finite instruction machine with finite memory (Finite State Automata)



Finite instruction machine with unbounded memory (Turing machine)



From finite automata to Turing Machines

1. A Turing machine can both write on the tape and read from it.
2. The read-write head can move both to left and right.
3. The tape is infinite.
4. Once accept/reject states are reached, the computation terminates at once.

Example

Consider the example language $B = \{w\#w \mid w \in \{0,1\}^*$

- Is it regular?
- How does one check if a given (really long) string is of this form?

Example

Consider the example language $B = \{w\#w \mid w \in \{0,1\}^*$

– Is it regular?
– How does one check if a given (really long) string is of this form?

1. Scan the input once to make sure it does contain a single $\#$. Else reject.
2. Repeatedly do: Mark the first "unmarked" symbol from beginning and zig-zag across the tape to its corresponding position on the other side of $\#$ symbol to check if it is the same. If not, reject.
3. Stop when there are no unmarked symbols remaining to the left of $\#$. If there are still symbols left on right, then reject. Else accept.

Example

Consider the example language $B = \{w\#w \mid w \in \{0,1\}^*$

- Is it regular?
- How does one check if a given (really long) string is of this form?

1. Scan the input once to make sure it does contain a single $\#$. Else reject.
2. Repeatedly do: Mark the first "unmarked" symbol from beginning and zig-zag across the tape to its corresponding position on the other side of $\#$ symbol to check if it is the same. If not, reject.
3. Stop when there are no unmarked symbols remaining to the left of $\#$. If there are still symbols left on right, then reject. Else accept.

In general, we use the finite control to process the symbols that are being read on tape. How does one formalize this?

Formal definition of a Turing Machine

Definition

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. q_0 is the start state
5. q_{acc} is the accept state
6. q_{rej} is the reject state
7. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.

Formal definition of a Turing Machine

Definition

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. q_0 is the start state
5. q_{acc} is the accept state
6. q_{rej} is the reject state
7. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.

A configuration is a snapshot of the system during computation. Thus, it can be described by the current state, the tape contents and the current head location.

Computation of a Turing Machine

Notation for configurations

Succinctly, we write $C = uqv$, if uv is the tape content, current state is q and head is at start of v.

We define C_1 yields C_2 if the TM can move from C_1 to C_2 in one step:

- left move: $u\ a\ q_i\ b\ v$ yields $u\ q_j\ a\ c\ v$ if $\delta(q_i, b) = (q_j, c, L)$
- right move: $u\ a\ q_i\ b\ v$ yields $u\ a\ c\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$
- left-end: $q_i\ b\ v$ yields (1) $q_j\ c\ v$ if transition is left moving or (2) $c\ q_j\ v$ if it is right moving
- right-end: assume that $u\ a\ q_i$ is the same as $u\ a\ q_i\ \sqcup$ as tape has blanks to the right.

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.

- A TM M accepts input word w if there exists a sequence of configurations $C_1, C_2, \ldots, C_k$ such that
  - C_1 is the start configuration
  - each C_i yields C_{i+1}
  - C_k is an accepting configuration
- Language of TM M, denoted $L(M)$ is the set of strings accepted by it.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing decidable or recursive

A language is decidable (or recursive) if there is a Turing machine accepting it, which has the additional property that it halts on all possible inputs.

Every decidable language is Turing recognizable, but is the converse true?

Examples of Turing Machines

Construct a TM for the language considered earlier $B = \{w\#w \mid w \in \{0,1\}^*\}$

- $Q = \{q_1, q_{acc}, q_{rej}, \dots, \}$
- $\Sigma = \{0, 1, \#\}$, $\Gamma = \{0, 1, \#, \sqcup, \dots, \}$
- start, accept, reject are q_1, q_{acc}, q_{rej}
- $\delta =?$

see board.

Examples of Turing Machines

Construct a TM for the language considered earlier $B = \{w\#w \mid w \in \{0,1\}^*\}$

- $Q = \{q_1, q_{acc}, q_{rej}, \ldots,\}$
- $\Sigma = \{0, 1, \#\}, \Gamma = \{0, 1, \#, \sqcup, \ldots,\}$
- start, accept, reject are q_1, q_{acc}, q_{rej}
- $\delta = ?$

see board.

1. Scan input to make sure it does contain a single #. Else reject.
2. Repeatedly do: Mark first “unmarked” symbol from left and zig-zag across tape to its corresponding position on the other side of # symbol to check if it is the same. If not, reject.
3. Stop when there are no unmarked symbols remaining to the left of #. If there are still symbols left on right, then reject. Else accept.

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n,0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape

Same as writing $B = \{a^m b^n c^{\max\{m-n,0\}} \mid m,n \geq 0\}$ and asked for a TM which accepts language B.

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n, 0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape

- M repeatedly replaces leading 0 by blank, then searches right for a 1 followed by 0 and changes 0 to 1.
- Then, M moves left until it encounters a blank and repeats this cycle.
- The repetition ends if
  1. searching right for 0, M encounters a blank.
     - Then all n 0's in 0^m10^n have been changed to 1 and $n+1$ of m 0's changed to blank. M replaces $n+1$ 1's by a 0 and n blanks leaving $m-n$ 0's on the tape.
  2. beginning the cycle, M cannot find a 0 to change to a blank, because first m 0's have already been changed.
     - Then $n \geq m$, so $\max\{m-n, 0\} = 0$. M replaces remaining 1's, 0's by blanks.

Variants of a TM

- Multi-tape TMs.
- Non-deterministic TMs
- Multi-head TMs
- Double side infinite tape TMs
- ...

What are the relative expressive powers? Do we get something strictly more powerful than standard TMs?

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

Theorem

Every multi-tape TM has an equivalent single-tape TM.

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank. Formally $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

Theorem

Every multi-tape TM has an equivalent single-tape TM.

(why not use the finite control to keep track of the k-many heads?)

Running time of a Turing Machine

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.

Running time of a Turing Machine

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.
- The time complexity of M is the function $T(n)$ that is the maximum, over all inputs w of length n, of the running time of M on w.

Running time of a Turing Machine

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.
- The time complexity of M is the function $T(n)$ that is the maximum, over all inputs w of length n, of the running time of M on w.
- We will be eventually more interested in TMs that halt on all inputs, and in particular have a polynomial time complexity $T(n)$.

Running time of a Turing Machine

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.
- The time complexity of M is the function $T(n)$ that is the maximum, over all inputs w of length n, of the running time of M on w.
- We will be eventually more interested in TMs that halt on all inputs, and in particular have a polynomial time complexity $T(n)$.

- What is the relation between running times of a 1 and multi-tape TM?

Running time of a Turing Machine

Running time of a TM

- The running time of a TM is the number of steps it makes before halting.
- So, if the TM doesnt halt, the running time is infinite.
- The time complexity of M is the function $T(n)$ that is the maximum, over all inputs w of length n, of the running time of M on w.
- We will be eventually more interested in TMs that halt on all inputs, and in particular have a polynomial time complexity $T(n)$.

- What is the relation between running times of a 1 and multi-tape TM?

The difference between polynomial time and higher growth rates in running time is really the divide between what we can “solve” by computer and what in practice is not solvable.

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

Every non-det TM is equivalent to a det TM

Proof idea:

1. View NTM N's computation as a tree.
2. explore tree using bfs and for each node (i.e., config) encountered, check if it is accepting.

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

Every non-det TM is equivalent to a det TM

Proof idea:

1. View NTM N's computation as a tree.
2. explore tree using bfs and for each node (i.e., config) encountered, check if it is accepting.

What is the time-complexity blow-up?

Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).

Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).
- Thus, though there are several computational models, the class of algorithms (or procedures) they describe is the same.

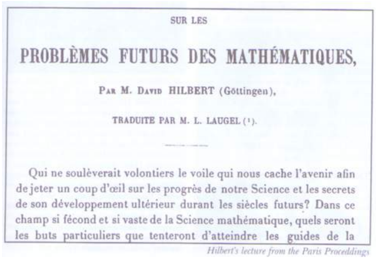
Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).
- Thus, though there are several computational models, the class of algorithms (or procedures) they describe is the same.
- Can do everything a computer can do and vice versa. But takes a lot more time. Is not practical and indeed its not what is implemented in today’s computer. After all who wants to write 100 lines to do subtraction or check something that a 4-year old can do?

Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).
- Thus, though there are several computational models, the class of algorithms (or procedures) they describe is the same.
- Can do everything a computer can do and vice versa. But takes a lot more time. Is not practical and indeed its not what is implemented in today's computer. After all who wants to write 100 lines to do subtraction or check something that a 4-year old can do?
- So then again, why Turing? Why is the top-most nobel-equivalent award in computer science called Turing award and not Bill Gates award?

Turing Machines: a lesson from history



SUR LES

PROBLÈMES FUTURS DES MATHÉMATIQUES,

PAR M. DAVID HILBERT (Göttingen),

TRADUITE PAR M. L. LAUGEL (¹).

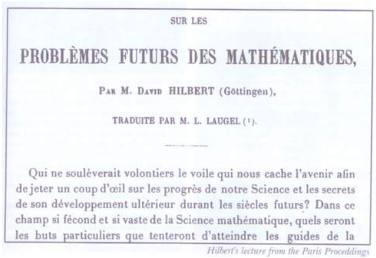
Qui ne soulèverait volontiers le voile qui nous cache l'avenir afin de jeter un coup d'œil sur les progrès de notre Science et les secrets de son développement ultérieur durant les siècles futurs? Dans ce champ si fécond et si vaste de la Science mathématique, quels seront les buts particuliers que tenteront d'atteindre les guides de la

Hilbert's lecture from the Paris Proceedings

Hilbert's problems

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no. of operations) to test if a given polynomial has integral roots.

Turing Machines: a lesson from history



SUR LES

PROBLÈMES FUTURS DES MATHÉMATIQUES,

PAR M. DAVID HILBERT (Göttingen),

TRADUITE PAR M. L. LAUGEL (¹).

Qui ne soulèverait volontiers le voile qui nous cache l'avenir afin de jeter un coup d'œil sur les progrès de notre Science et les secrets de son développement ultérieur durant les siècles futurs? Dans ce champ si fécond et si vaste de la Science mathématique, quels seront les buts particuliers que tenteront d'atteindre les guides de la

Hilbert's lecture from the Paris Proceddings

Hilbert's problems

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no. of operations) to test if a given polynomial has integral roots.
- Now we know that no such algorithm exists. But how to prove this without a mathematical definition of an algorithm?

The Church-Turing Thesis

Alonso Church
(1903–1995)

Alan Turing
(1912–1954)

The Church-Turing Thesis

- The Church-Turing Thesis provides the definition of algorithm necessary to resolve Hilbert's tenth problem.

The Church-Turing Thesis

Our intuitive understanding of algorithms coincides with Turing machine algorithms.

The Church-Turing Thesis

- The Church-Turing Thesis provides the definition of algorithm necessary to resolve Hilbert's tenth problem.

The Church-Turing Thesis

Our intuitive understanding of algorithms coincides with Turing machine algorithms.

- Thus, Turing machines capture our intuitive idea of computation.
- Indeed, we are more interested in algorithms themselves. Once we believe that TMs precisely capture algorithms, we will shift our focus to algorithms rather than low-level descriptions of TMs.

More on the Church-Turing Thesis

Lambda Calculus

- In 1936, Church introduced Lambda Calculus as a formal description of all computable functions.
- Independently, Turing had introduced his A-machines in 1936 too.
- Turing also showed that his A-machines were equivalent to Lambda Calculus of Church.

- So, can a Turing machine do everything? In other words are there algorithms to solve every question. Godel's incompleteness result asserts otherwise.
- If there is TM solving a problem, does there exist an equivalent TM that halts?

What else did Turing do?

– Alan Turing characterized computable functions by building a machine. Though theoretical this gave rise to the idea of computers.
– But Turing also worked on ideas and concepts that later made profound impact in AI.